Hyper-parameter Tuning with Python

Learning Outcomes

By the end of this topic, you will have achieved the following learning outcomes:

- I can understand the concept of hyper-parameter tuning in solving problems.
- I can apply hyper-parameter tuning techniques using grid search and randomized search to optimize models.
- I can differentiate between different hyper-parameter tuning techniques.
- I can understand when to apply different hyper-parameter tuning techniques.

"I am fascinated with the idea of the brand becoming a platform, not just for self-expression, but to create unique bespoke products in a one-to-one relationship that helps them feel empowered and strong and able to express."

Overview

Reading

Hyperparameter tuning refers to the optimization of the hyper-parameters of a model. In our case, hyperparameters would be all the parameters of a model which are not updated during the learning and are used to configure either the model (e.g. size of the hashing space, number of decisions trees and their depth, number of layers of a deep neural network, etc.)

While performing hyperparameter tuning, we look and set the hyper-parameters that lead to the lowest error on the validation set.

Hyperparameters vs. Model parameters

- **Hyperparameters**
    - These are the adjustable parameters that must be tuned in order to obtain a model with optimal performance.

- **Model parameters**
    - These are the parameters in the model that must be determined using the training data set. They are the fitted parameters.

Hyperparameters might address model design questions, in a decision tree, such as:

- What should be the maximum depth allowed for my decision tree?
- What should be the minimum number of samples required at a leaf node in my decision tree?
- How many trees should I include in my random forest?
- What should I set my learning rate for gradient descent?

Hyperparameters can make a big difference in the performance of a machine learning model.

Examples of Hyperparameters

1. **Logistic Regression Classifier**

   LogisticRegression(C=1000.0, random_state=0)

   C is the inverse of regularization strength, and ***random_state*** is the seed of the pseudo-random number generator to use when shuffling the data.

2. **KNN (k-Nearest Neighbors) Classifier**

   KNeighborsClassifier(n_neighbors=5, p=2, metric='minkowski')

   n_neighbors is the number of neighbours to use, ***p*** is the power parameter for the Minkowski metric. When p = 1, this is equivalent to using ***manhattan_distance***, and ***euclidean_distance*** for p = 2.

3. **Support Vector Machine Classifier**

   SVC(kernel='linear', C=1.0, random_state=0)

   kernel specifies the kernel type to be used in the algorithm, for example, kernel = ‘linear’, for linear classification, or kernel = ‘rbf’ for non-linear classification. ***C*** is the penalty parameter of the error term, and ***random_state*** is the seed of the pseudo-random number generator used when shuffling the data for probability estimates.

4. **Decision Tree Classifier**

DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)

criterion is the function to measure the quality of a split, ***max_depth*** is the maximum depth of the tree, and ***random_state*** is the seed used by the random number generator.

5. **Principal Component Analysis**

   PCA(n_components = 4)

   n_components is the number of components to keep. If n_components is not set all components are kept.

If you would like to investigate the parameters of a particular model, you can refer to its documentation.

Upon model evaluation, it is important these hyperparameters be fine-tuned in order to obtain the model with the highest quality.

The hyper-parameter optimization algorithms can be categorized into the following:

1. **Manual Search**
   - While performing this type of hyper-parameter tuning, we choose model hyperparameters based on our judgment, experience or domain expertise.
   - We train the model, evaluate its accuracy and start the process again until a satisfactory accuracy is achieved.
   - This process can be time-consuming and may not necessarily yield the best results since getting the right combination of hyper-parameters is difficult.
   - However, with experience and extensive domain expertise, it becomes easier to achieve our goal.
2. **Grid Search**
   - Grid search entails building a model for each possible combination of all of the hyperparameter values provided, evaluating each model, and selecting the one which produces the best results.
   - It is the most expensive method in terms of total computation time.
3. **Random Search**
   - Random search is a slight variation on grid search. Instead of searching over the entire grid, random search only evaluates a random sample of points on the grid.
   - The number of search iterations is usually set based on the allocated time and resources. This makes the random search a lot cheaper than grid search.

4. **Bayesian Optimisation**
    - Bayesian optimisation works by getting an optimal set of parameters by taking into account information on the hyperparameter combinations it has seen. It does this by means of bayesian reasoning.
    - Bayesian optimisation components:
        - **Choosing the Search Space to Sample Parameters From**
            - Bayesian Optimisation operates along with probability distributions for each parameter that it will sample from.
        - **The Objective Function**
            - The objective function serves as the main evaluator of hyperparameter combinations. It simply takes in a set of hyperparameters and outputs a score that indicates how well a set of hyperparameters performs on the validation set.
        - **The Surrogate and Selection Function**
            - Both work together to propose those parameters of which it believes will bring the highest accuracy on the objective function. The surrogate function can be interpreted as an approximation of the objective function. It is used to propose parameter sets, to the objective function that likely yield an improvement in terms of accuracy score.

Choosing the right hyper-parameter optimization algorithm depends on the research problem, the computing infrastructure and the associated computing budget, etc.

It's usually advisable first to look into these aspects, and understanding whether your technical criteria have been achieved by the default values of the hyperparameter.

References

You can also use the following these resources for your further reading.

1. Introduction to Hyper-parameter tuning: [Link]
2. Demystifying Hyper-Parameter tuning [Link]
3. Parameter Optimisation: [Link]
4. Common problems in Hyperparameter Optimisation [Link]